TP12 : Bin packing

Objectifs du TP :

- Résoudre un problème NP-complet de manière approchée avec garantie sur la qualité du résultat.
- Analyser un algorithme d'approximation à facteur constant.
- Montrer l'inexistence de bonnes approximations pour un problème NP-complet.

Partie 0 *Problème du bin packing*

On s'intéresse au problème d'optimisation BIN PACKING défini comme suit :

$$\begin{cases} \textbf{Entrée} : n \in \mathbb{N}^\star \text{ objets de volumes } v_1, ..., v_n \text{ et un volume maximal } V \in \mathbb{N}^\star. \\ \textbf{Solution} : \text{Un entier } m \text{ et une fonction } f : [\![1, n]\!] \to [\![1, m]\!] \text{ tels que pour tout } i \in [\![1, m]\!], \sum_{f(v_j)=i} v_j \leq V. \\ \textbf{Optimisation} : \text{Minimiser } m. \end{cases}$$

Autrement dit, on cherche à ranger les n objets dans m boîtes de volume maximal V de telle sorte à ce que la somme des volumes des objets dans une boîte n'excède jamais V et en utilisant le nombre de boîtes minimal. Un exemple concret dans lequel ce problème intervient est le suivant. On dispose de trains de marchandises pouvant chacun contenir un volume fixé et de marchandises de volumes divers à acheminer : quel est le nombre minimal de trains à utiliser pour pouvoir déplacer toute la marchandise ?

0) Résoudre BIN PACKING de manière exacte lorsque $V = 10$ et les volumes des objets sont $2, 5, 4, 7, 1, 3, 8$.

Le sujet est décomposé en une partie pratique, dans laquelle on vous demande d'implémenter trois algorithmes d'approximation pour ce problème, et d'une partie théorique d'analyses. Durant l'heure de TP, concentrez vous d'abord sur la partie programmation.

Partie 1 *Algorithmes d'approximation pour BIN PACKING*

On supposera à partir de maintenant que le volume de chacun des objets est inférieur au volume maximal des boîtes ; sans quoi, aucun rangement n'est possible.

1. Décrire un type box permettant de modéliser une boîte. On demande à ce que le calcul du volume libre dans une boîte puisse se faire en temps constant et à ce qu'on puisse facilement y rajouter un objet.

On propose dans cette partie trois algorithmes gloutons d'approximation pour BIN PACKING. Le premier algorithme sera appelé *next-fit*. Son principe est le suivant : il maintient à jour une boîte courante. Pour chaque objet, *next-fit* détermine s'il peut rentrer dans la boîte courante : si oui, il l'y place, si non, il ferme définitivement la boîte courante, ouvre une nouvelle boîte qui devient la nouvelle boîte courante et y place l'objet.

2. Ecrire une fonction `add_next : int -> int -> box list -> int -> box list`. Elle prend quatre arguments en entrée : le numéro i d'un objet, son volume, une liste de boîtes L (on suppose sans le vérifier que la capacité maximale de chaque boîte est respectée) et le volume maximal de chacune de ces boîtes. Elle renvoie une liste correspondant à la nouvelle répartition des objets dans les boîtes après ajout de l'objet i selon la stratégie *next-fit*. *Indication : quel élément de L est-il judicieux de considérer comme étant la boîte courante ?*
3. En déduire une fonction `next_fit : int -> int array -> int` prenant en entrée le volume maximal des boîtes, un tableau de taille n contenant en case i le volume du i-ème objet et renvoyant le nombre de boîtes nécessaires pour stocker les n objets selon l'algorithme d'approximation *next-fit*.

Le deuxième algorithme utilise l'heuristique *first-fit*. Il consiste à maintenir une liste (initialement vide) de boîtes $B_1, ..., B_k$. Il considère ensuite chaque objet et le place dans la première boîte qui peut le contenir en commençant par B_1. S'il ne rentre dans aucune boîte, il créé la boîte B_{k+1} et y place l'objet.

4. Ecrire une fonction récursive `add_first : int -> int -> box list -> int -> box list` ayant la même spécification que `add_next` mais utilisant la stratégie *first-fit*.

5. En déduire une fonction `first_fit : int -> int array -> int` de même spécification que `next_fit` mais utilisant la stratégie *first-fit*.

Le dernier algorithme qu'on étudie dans ce TP est FFD (first-fit decreasing). Le principe est le même que pour l'algorithme first-fit à cette différence près que les objets sont considérés par ordre de volume décroissant.

6. Ecrire une fonction `ffd : int -> int array -> int` de même spécification que `first_fit` mais utilisant cette nouvelle stratégie gloutonne. On pourra relire la documentation concernant la fonction `Array.sort`.

7. Vérifier que les fonctions `next_fit`, `first_fit` et `ffd` renvoient un résultat cohérent sur l'entrée décrite à la question 0. En appliquant ces fonctions à quelques entrées de plus grande taille générées aléatoirement, ordonner empiriquement les facteurs d'approximation de *next-fit*, *first-fit* et FFD.

Partie 2 *Analyses choisies*

Cette partie se subdivise en trois sous-parties :

- Preuve de la NP-complétude de BIN PACKING
- Analyse de la qualité de l'approximation donnée par l'algorithme next-fit et de sa complexité.
- Preuve de non approximabilité de BIN PACKING pour des facteurs d'approximation trop petits.

NP-complétude de BIN PACKING

On rappelle que le problème PARTITION suivant est NP-complet :

$$\begin{cases} \textbf{Entrée} : n \text{ entiers naturels } c_1, ..., c_n. \\ \textbf{Question} : \text{Y a-t-il un sous-ensemble } S \subset [\![1, n]\!] \text{ tel que } \sum_{i \in S} c_i = \sum_{i \notin S} c_i \ ? \end{cases}$$

8. Décrire le problème de décision BPD associé au problème d'optimisation BIN PACKING.

9. Montrer que BPD $\in$ NP.

10. Etant donnée une instance $c_1, ..., c_n$ de PARTITION, on construit l'instance suivante de BPD : les volumes sont $2c_1, ...2c_n$, le volume total est $\sum_{i=1}^{n} c_i$ et le seuil est égal à 2. Montrer que cette transformation montre que PARTITION se réduit polynomialement en BPD et conclure.

Analyse de *next-fit*

On note m le nombre de boîtes déterminé via la stratégie *next-fit* sur une instance donnée de BIN PACKING et $m^\star$ le nombre optimal de boîtes pour cette même instance.

11. Montrer que la somme des volumes occupés par des objets de deux boîtes consécutives selon la stratégie *next-fit* est strictement supérieure à V.

12. Montrer que $m^\star V \geq \sum_{i=1}^{n} v_i$.

13. En déduire que *next-fit* est une 2-approximation de BIN PACKING.

14. Déterminer la complexité de *next-fit* et commenter.

15. Existe-t-il un $\alpha < 2$ tel que *next-fit* soit une α-approximation de BIN PACKING ?

Remarque : Il est évident que first-fit utilise moins de boîtes que next-fit, par conséquent ce second algorithme est aussi une 2-approximation de BIN PACKING. On peut en fait montrer que first-fit est une 1.7-approximation de BIN PACKING.

Non approximabilité

Soit $\varepsilon > 0$ et supposons qu'il existe un algorithme polynomial qui soit une $(3/2 - \varepsilon)$-approximation de BIN PACKING.

16. En s'inspirant de la réduction de la question 10, montrer que dans ces conditions il existe un algorithme polynomial permettant de résoudre PARTITION.

Remarque : Si $P \neq NP$ on vient donc de montrer qu'il n'existe pas d'α-approximation de BIN PACKING pour $\alpha < 3/2$. Il se trouve que l'algorithme FFD permet de trouver un nombre de boîtes m tel que $m \leq 3m^\star/2 + 1$ ce qui en fait un très bon algorithme d'approximation pour BIN PACKING.